我们为什么要拥抱 React Hook

Hook 是一项新的功能提案，可以让你在不编写类的情况下使用状态（state）和其他 React 功能。

```
import { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

这个 useState 新功能将是我们学习的第一个“钩子”，但这个例子仅仅是个预告。即便你对它没感觉也不用担心！

在本页面上，我们将继续解释为什么我们要将 Hook 添加到 React 中，以及它们如何帮助你编写出色的应用。

没有重大的变化

在我们继续之前，请注意 Hook 是：

- **完全可选择引入。** 你无需重写任何现有代码，就能在一些组件里尝试使用 Hook。但如果你不想，你不必现在学习或使用 Hook。
- **100% 向后兼容。** Hook 不包含任何重大的更改。
- **现在可用。** Hook 目前处于 alpha 版本，我们希望在收到社区反馈后把它们包含在 React 16.7 中。

没有把类从 React 中移除的计划。Hook 不会取代你对 React 概念的理解。 相反地，Hook 为你已知的 React 概念（props、state、context、refs 和 lifecycle）提供了更直接的 API。并且稍后我们还将演示，Hook 还提供了一种组合它们新的强大的方式。

动机

Hook 解决了我们在过去五年时间里编写和维护数以万计 React 组件时遇到的各种看似不相关的问题。无论你是在学习 React，还是日常使用，甚至说是喜欢使用具有类似组件模型的其他库，你都有可能注意到这些问题。

在组件之间重用带状态逻辑很困难

React 没有提供把可重用行为“附加”到组件上的方法（例如，把它关联到 store 里）。如果你已经使用了 React 一段时间，你可能会熟练使用渲染属性和高阶组件的模式尝试解决这个问题。但这些模式需要你在使用它们的时候对组件进行重构，这可能会很麻烦并且使代码更难以跟踪。如果你看一下 React DevTools 里的典型 React 应用程序，你也许会发现一个由提供者、消费者、高阶组件、渲染属性和其他抽象层包裹起来的“包装地狱”组件。虽然我们可以在 React DevTools 中过滤它们，但这里引出了一个更深层次的基本问题：React 需要一个更好的分享带状态逻辑的原语。

使用 Hook，你可以从一个组件中导出带状态逻辑，以便它可以单独测试和复用。**Hook 允许你在不改变组件层次结构的情况下复用带状态逻辑。** 这样就可以轻松地在多个组件之间或者社区里共享 Hook。

我们将在编写自定义钩子里进行更多的讨论。

复杂的组件变得难以理解

我们经常不得不维护从简单开始，但后来变成混杂着带状态逻辑和副作用无法管理的组件。每个生命周期方法通常都包含了一堆不相关的逻辑。例如，组件也许要在 componentDidMount 和 componentDidUpdate 方法里请求数据。但是，同样是在 componentDidMount 方法，可能会包含不相关的设置事件监听的逻辑，还得在 componentWillUnmount 里清除。本该一起更改的相关联代码被拆分，而完全不相关的代码却最终组合到一个方法里。这就太容易引入 bug 和导致不一致了。

很多情况下不可能把这些组件拆分得更小，因为到处都有带状态的逻辑。而且测试它们也很困难。这就是许多人更喜欢将 React 与单独的状态管理库相结合的原因之一。但是，这通常会引入太多的抽象(概念)，使得你在不同的文件之间跳转，同时让重用组件变得更加困难。

为了解决这个问题，**Hook 允许你基于相关联的部分(例如设置订阅或获取数据)把一个组件拆分成较小的函数**，而不是基于生命周期函数强制拆分。你还可以选择使用 reducer 管理组件的本地状态，以使其更具可预测性。

我们将在使用效果 Hook 里更多地讨论这个问题。

类（Class）混淆了人类和机器

通过我们的观察，发现类是学习 React 最大的障碍。你必须理解 this 在 JavaScript 中是怎么工作的，大多数语言中它的工作方式有很大不同。你必须记住绑定事件处理程序。如果没有不稳定的语法提案，代码就非常冗长。人们可以很好地理解属性，状态和自上而下的数据流，但仍然很艰难地与类作斗争。React 中的函数和类组件之间的区别以及何时使用哪种组件，即使在经验丰富的 React 开发人员之间也会引发分歧。

此外，React 已经推出大概五年时间了，并且我们希望确保它在未来的五年里还保持相关性。就像 Svelte，Angular，Glimmer 和其他人表明的那样，提前编译组件未来有很大的潜力。特别是在它不局限于模版的情况下。目前，我们已经使用 Prepack 做了组件折叠的实验，并且我们已经看到了有前景的早期结果。但是，我们发现类组件可能会引发无意识的模式使得这些优化回退到较慢的路径上。类也给今天的工具提出了问题。例如，类不能很好地压缩，并且它们使得热更新加载变得片状和不可靠。我们希望提供一种 API，使代码更可能的留在可优化的路径上。

为了解决这个问题，**Hook 允许你在没有类的情况下使用更多 React 的功能。**概念上来说，React 组件一直是更接近于函数的。Hook 拥抱函数，但不会牺牲掉 React 实际的精神。Hook 提供了对命令式逃生舱口的访问，并且不需要你学习复杂的函数式或反应式编程技术。

逐步采用策略

TLDR：没有从 React 中移除类的计划

我们知道 React 开发者专注于发布产品，没有时间研究正在发布的每个新 API。Hook 是很新的，在考虑学习或采用它们之前等待更多的示例和教程可能会更好。

我们也理解为 React 添加新原语的标准非常高。对于好奇的读者来说，我们已经事先准备了一个详细的 RFC，里面有更多深入细节的动机，并提供有关特定设计决策的额外视角和相关领先技术。

至关重要的是，Hook 和现有代码是并行工作的，所以你可以逐步采用它们。我们正在分享这个实验性的 API，为了是从社区中那些有兴趣塑造 React 未来的人那里得到早期反馈 —— 然后我们会在公开场合迭代 Hook。

最后，别急着迁移到 Hook。我们建议避免任何“重大改写”，特别是对于现有复杂的类组件。开始“考虑 Hook”需要一点心理上的转变。根据我们的经验，

最好先在新的和非相关的组件里练习使用 Hook，并确保团队中的每个人都对它们感到满意。